PCS 2428 / PCS 2059
Inteligência Artificial

Prof. Dr. Jaime Simão Sichman
Profa. Dra. Anna Helena Reali Costa

Busca Local e
Problemas de Otimização

Agenda

Classe de problemas de interesse

- Em vários problemas a própria descrição de estado contém toda informação relevante para a solução e o caminho ao estado-objetivo **não** interessa:
  – Ex: problema das 8 rainhas, projeto de circuitos integrados, escalonamento, problemas de roteamento, de otimização de redes de telecomunicação, etc.
    ➔ **problemas de otimização**
- Buscas Locais (ou de melhorias iterativas) operam num único estado e movem-se para a vizinhança deste estado.

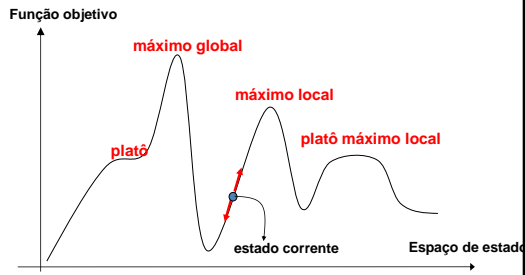
Busca Local

- A idéia é começar com o *estado inicial* (configuração completa, solução aceitável), e melhorá-lo iterativamente.
- Visualização:
  – Os estados (≡ solução) estão representados sobre uma **superfície** (gráfico);
  – A **altura** de qualquer ponto na superfície corresponde à **função de avaliação** do estado naquele ponto;
  – O algoritmo se "**move**" pela superfície em **busca de pontos mais altos** (melhor avaliação do estado);
  – O ponto mais alto (**máximo global**) corresponde à **solução ótima.**

Exemplo de Espaço de Estados

Busca Local

- Esses algoritmos armazenam apenas o estado atual (baixo uso de memória), e não vêem além dos vizinhos imediatos do estado.
- Apesar destas restrições, muitas vezes são os melhores métodos para tratar problemas reais muito complexos (espaço contínuo).

Espaço de estados unidimensional

Tipos de Busca local

- *Hill-Climbing*: Subida pela Encosta mais Íngrime ou Busca Local Gulosa
  - só faz modificações que melhoram o estado atual.
- *Simulated Annealing*: Têmpera Simulada
  - pode fazer modificações que pioram o estado no momento, para possivelmente melhorá-lo no futuro.
- *Local beam search*: Busca em feixe local
  - Mantém k estados em vez de um único.
- Algoritmos genéticos (GA)
  - É uma busca **subida pela encosta**, estocástica, na qual uma grande população de estados é mantida e novos estados são gerados por mutação ou cruzamento.

8

Subida da Encosta

- O algoritmo *não* mantém uma árvore de busca:
  - guarda apenas o estado atual e sua avaliação
  - É simplesmente um ciclo que move o estado (solução) na direção **crescente** da função de avaliação (*muda o estado para o melhor vizinho*).

9

Subida da Encosta

Hill-climbing (or gradient ascent/descent)

"Like climbing Everest in thick fog with amnesia"

```
function HILL-CLIMBING(problem) returns a state that is a local maximum
   inputs: problem, a problem
   local variables: current, a node
                    neighbor, a node
   current ← MAKE-NODE(INITIAL-STATE[problem])
   loop do
      neighbor ← a highest-valued successor of current
      if VALUE[neighbor] ≤ VALUE[current] then return STATE[current]
      current ← neighbor
   end
```

At each step, move to a neighbor of higher value
in hopes of getting to a solution having the highest possible value

Can easily modify this for problems where we want to minimize rather than maximize

10

Subida da Encosta – Problemas

- Isso pode acarretar 3 tipos de problemas:
  1. Máximos locais
  2. Planícies (platôs)
  3. Encostas e picos: somente poucos vizinhos podem melhorar a solução (difícil de encontrá-los)
- Nestes casos, o algoritmo chega a um ponto de onde não faz mais progresso.

11

Subida da Encosta – Problemas

- *Solução: reinício aleatório (random restart)*
  - O algoritmo realiza uma série de buscas a partir de estados iniciais gerados aleatoriamente.
  - Cada busca é executada
    - até que um número máximo estipulado de iterações seja atingido, ou
    - até que os resultados encontrados não apresentem melhora significativa.
  - O algoritmo escolhe o melhor resultado obtido com as diferentes buscas (diferentes reinícios).

12

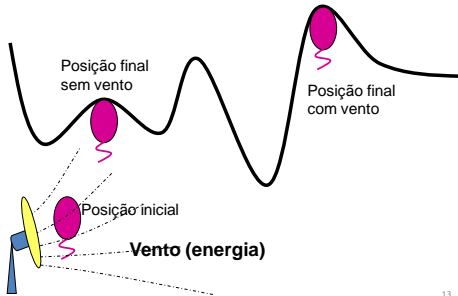
Têmpera Simulada

- Semelhante à Subida pela Encosta, porém oferece meios para escapar de máximos locais.

Posição final sem vento

Posição final com vento

Posição inicial

Vento (energia)

Têmpera Simulada

- Têmpera simulada ou Arrefecimento simulado é uma *meta-heurística* para otimização que consiste numa técnica de busca local probabilística, e se fundamenta numa metáfora de um processo térmico de metalurgia, dito *recozimento:*
  - Passo 1: a temperatura do sólido é aumentada para um valor máximo no qual ele se funde;
  - Passo 2: o resfriamento deve ser realizado lentamente até que o material se solidifique.

  Isto provoca que os átomos desse material ganhem energia para se movimentarem livremente e, ao arrefecer de forma controlada, dar-lhes uma melhor hipótese de se organizarem numa configuração com menor energia interna, para ter, como resultado prático, uma redução dos defeitos do material.

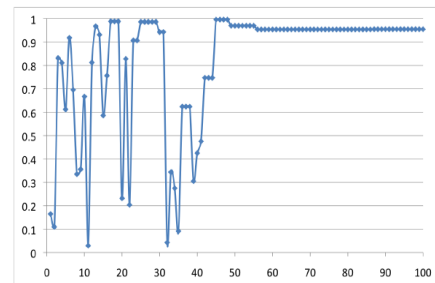
Têmpera Simulada

Simulated annealing

Idea: escape local maxima by allowing some "bad" moves
but gradually decrease their size and frequency

```
function SIMULATED-ANNEALING(problem, schedule) returns a solution state
   inputs: problem, a problem
           schedule, a mapping from time to "temperature"
   local variables: current, a node
                    next, a node
                    T, a "temperature" controlling prob. of downward steps
   current ← MAKE-NODE(INITIAL-STATE[problem])
   for t ← 1 to ∞ do
       T ← schedule[t]
       if T = 0 then return current
       next ← a randomly selected successor of current
       ΔE ← VALUE[next] − VALUE[current]
       if ΔE > 0 then current ← next
       else with probability e^(ΔE/T), set current ← next
```

Têmpera Simulada – Exemplo

100 iterations, each state is a number $x \in [0,1]$, initial state is $x = 0$, $\text{VALUE}(x) = x^2$, all states are neighbors, $schedule[i] = 10 \times 0.9^i$

Busca em Feixe Local

- Começa com **k** estados gerados aleatoriamente.
- Em cada passo, são gerados <u>todos</u> os sucessores de <u>todos</u> os **k** estados.
- Se um dos sucessores for o objetivo, o algoritmo para; caso contrário, escolhe os **k** melhores sucessores a partir da lista <u>completa</u>.
  - Note que isso NÃO corresponde à execução de k reinícios aleatórios em paralelo da busca local subida da encosta (*random start*)!
  - Note que sempre somente **k** estados são considerados como estados atuais na busca.

Busca em Feixe Local

Local beam search

```
function BEAM-SEARCH(problem, k) returns a solution state
   start with k randomly generated states
   loop
       generate all successors of all k states
       if any of them is a solution then return it
       else select the k best successors
```

Computação Evolucionária

- Toda tarefa de busca e otimização possui vários componentes, entre eles:
  - espaço de busca, onde são consideradas todas as possibilidades de solução de um determinado problema e
  - função de avaliação (ou função de custo), uma maneira de avaliar os membros do espaço de busca.

  Existem muitos métodos de busca e funções de avaliação.
- As técnicas de busca e otimização tradicionais iniciam-se com um **único candidato** que, iterativamente, é manipulado utilizando algumas heurísticas (estáticas) diretamente associadas ao problema a ser solucionado.

Computação Evolucionária

- As técnicas de computação evolucionária operam sobre uma **população de candidatos em paralelo**.
  - Assim, elas podem fazer a busca em diferentes áreas do espaço de solução, alocando um número de membros apropriado para a busca em várias regiões.
- Os Algoritmos Genéticos (AGs) diferem dos métodos tradicionais de busca e otimização, principalmente em cinco aspectos:
  1. Trabalham com uma codificação do conjunto de parâmetros e não com os próprios parâmetros.
  2. Trabalham com uma população e não com um só elemento.
  3. Utilizam informações de custo ou recompensa.
  4. Utilizam regras de transição probabilísticas e não determinísticas.
  5. São baseados na técnica gerar-e-testar

Algoritmos Genéticos

- "Quanto melhor um indivíduo se adaptar ao seu meio ambiente, maior será sua chance de sobreviver e gerar descendentes": este é o conceito básico da evolução genética biológica, que inspira a classe de algoritmos AGs.
- Algoritmos Genéticos são algoritmos de **otimização global**, baseados nos mecanismos de seleção natural e da genética.
  - Empregam uma estratégia de busca paralela e estruturada, mas aleatória, que é voltada em direção ao reforço da busca de pontos de "alta aptidão", ou seja, pontos nos quais a função a ser minimizada (ou maximizada) tem valores relativamente baixos (ou altos).

Algoritmos Genéticos

- A modelagem de um problema AG envolve:
  - Codificação da solução
  - Função de avaliação
  - Função de aptidão (*fitness*)

Função de avaliação

- Função de avaliação:
  - Provém uma medida de desempenho com respeito a um conjunto particular de parâmetros.
  - Deve ser relativamente rápida, uma vez que, em cada iteração, cada membro da população é avaliado e recebe um valor de aptidão.
  - A avaliação de um membro (cromossomo) representando um conjunto particular de parâmetros é independente da avaliação de qualquer outro membro

Função de aptidão

- Função de aptidão:
  - Transforma a medida da função de avaliação em alocação de oportunidades reprodutivas.
  - É sempre definida de acordo com outros membros da atual população.
  - No algoritmo genético canônico, aptidão é definida como $fi_x = f(x)/f'$ onde $f(x)$ é a avaliação associada ao cromossomo **x** e f' é a soma da avaliação (ou média) de todos os membros da população.
  - A aptidão pode também ser associada à classificação de um cromossomo na população ou outras medidas.

Codificação

- O cromossomo deve, de algum modo, conter informação sobre a solução a qual representa.
- O modo mais usual de codificação é a seqüência binária. O cromossomo, então deve ser algo como:

Cromossomo 1	1101100100110110
Cromossomo 2	1101111000011110

- Cada bit na seqüência (gene) pode representar uma característica da solução ou a série como um todo pode representar um número.

25

Pseudo-algoritmo de um AG (canônico)

[Início] Gerar uma população aleatória com *n* cromossomos

Repetir até obter a solução (ou terminar)

geração

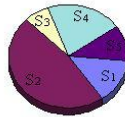
1. **[Avaliação]** Determinar ***f(x)*** e fi_x de cada cromossomo *x* na população.
2. **[Seleção]** Selecionar elementos para criar uma população intermediária //Quanto melhor o fi_x, maior chance de ser selecionado); uma porcentagem dos mais adaptados é mantida, enquanto os outros são descartados (mortos).
3. **[Recombinação (Crossover)]** Com uma probabilidade de recombinação, realizar uma recombinação sobre os pais para formar uma nova prole //Se não ocorrer recombinação, a prole é uma cópia exata dos pais.
4. **[Mutação]** Com uma probabilidade de mutação, realizar mutação sobre a nova prole em cada *gene* (posição no cromossomo).
5. **[Atualização da população]** Usar a nova população gerada para repetir os passos 1-5 do algoritmo

Retornar a melhor solução na população atual.

26

Seleção através da Roleta

- Os pais são selecionados de acordo com sua função de aptidão. Quanto melhor os cromossomos são, maior a chance de serem selecionados. Imagine uma **roleta** onde são colocados todos os cromossomos da população, o espaço ocupado depende do valor da sua função de aptidão, como mostra a figura:

Indivíduo S_i		$f(S_i)$	Aptidão Relativa
S1	10110	2.23	0.14
S2	11000	7.27	0.47
S3	11110	1.05	0.07
S4	01001	3.35	0.21
S5	00110	1.69	0.11

27

Elitismo

- Elitismo é o nome do método, aliado à seleção, que primeiro copia o melhor cromossomo (ou alguns melhores cromossomos) para a nova população. O processo restante é feito do modo clássico.
- O elitismo pode aumentar rapidamente o desempenho do AG porque evita que se perca a melhor solução encontrada até o momento.

28

Recombinação (Crossover)

- Seleciona genes a partir dos cromossomos pais e cria uma nova prole.
- O modo mais simples: 1. escolher aleatoriamente algum ponto (*locus* entre genes) no cromossomo, 2. tudo que estiver antes desse ponto será copiado do primeiro pai, 3. tudo que estiver depois será copiado do segundo pai.

Cromossomo 1	11011 | 00100110110
Cromossomo 2	11011 | 11000011110
Prole 1	11011 | 11000011110
Prole 2	11011 | 00100110110

- Existem outros meios de realizar recombinação, por exemplo, podemos escolher mais pontos de recombinação.
- Recombinações desenvolvidas para problemas específicos podem melhorar o desempenho do AG.

29

Mutação

- Mutação permite evitar que a população fique presa em um mínimo (máximo) local.
- A mutação altera aleatoriamente a nova prole. Na codificação binária, podemos mudar alguns bits de 1 para 0 e de 0 para 1:

Prole Original 1	**1110111000011110**
Prole Original 2	**1110110100110110**
Prole com Mutação 1	**1110011000011110**
Prole com Mutação 2	**11101101100110110**

- A mutação depende tanto da codificação como da recombinação.

30

Parâmetros - I

- **Probabilidade de recombinação:** indica o quão freqüente a recombinação é executada.
  - Se **não** ocorre recombinação, a prole é a cópia dos pais.
  - Se ocorre recombinação, a prole é formada por partes dos pais.
  - Se a probabilidade de recombinação é **100%**, então toda a prole será formada com recombinação.
  - Se for **0%**, a nova geração será formada por cópias exatas da população anterior
- A recombinação é realizada com a esperança de que os novos cromossomos tenham partes boas dos cromossomos anteriores e talvez sejam melhores que seus pais. Entretanto é aconselhável deixar parte da população sobreviver na geração seguinte.

Parâmetros - II

- **Probabilidade de mutação:** indica o quão freqüente partes dos cromossomos sofrerão mutações.
  - Se não ocorrer mutação (probabilidade de mutação = 0%), a prole não sofrerá mudanças após a recombinação.
  - Se ocorrer mutação, parte dos cromossomos será alterada.
  - Se a probabilidade de mutação for **100%,** todos os cromossomos serão alterados.
- A mutação é realizada para prevenir que o AG caia em um extremo local (mínimo ou máximo, depende das funções usadas), porém ela não deve ocorrer com muita freqüência, pois acarretaria em uma **busca aleatória** no espaço de soluções.

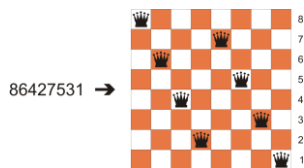
Parâmetros - III

- **Tamanho da população:** indica quantos cromossomos existem em uma população (em uma geração).
  - Se existirem muito poucos cromossomos, o AG terá poucas possibilidades de realizar a recombinação e apenas uma pequena porção do espaço de estados será explorada.
  - Se existirem muitos cromossomos, o AG ficará lento.
  - Pesquisas mostram que após algum limite (que depende principalmente da codificação e do problema) não é vantajoso aumentar o tamanho da população pois não tornará o algoritmo mais rápido.

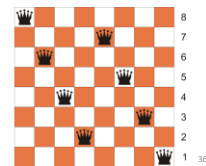
Parâmetros - IV

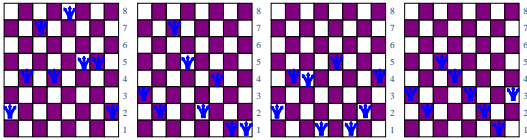
- **Intervalo de Geração**. Controla a porcentagem da população que será substituída durante a próxima geração.
  - Com um valor alto, a maior parte da população será substituída, mas com valores muito altos pode ocorrer perda de estruturas de alta aptidão.
  - Com um valor baixo, o algoritmo pode tornar-se muito lento.

Exemplo: 8 Rainhas

- **Codificação**
  - Cromossomos compostos por 8 números (genes), a posição do gene indica a coluna, o valor do número indica a linha

Exemplo: 8 Rainhas

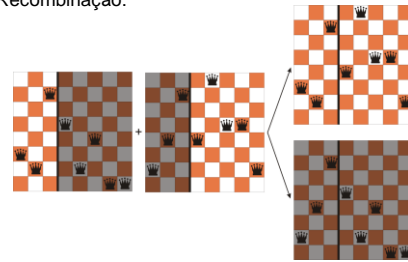
- Função de avaliação:
  - f(x) =número de pares que **não** se atacam
  - Na solução: 1-2,1-3, ...,1-8, 2-3,...,2-8, 3-4,..., 3-8,..., 6-7, 6-8, 7-8 = 28 pares
- Função de aptidão:
  - $fi_x = f(x)/\Sigma f(x)$ [%]

x: 1-8 em ataque ➔
➔ f(x) = 27

Exemplo: 8 Rainhas

- Considere a seguinte população inicial:

1. Determine a codificação de cada cromossomo.
2. Determine a população intermediária considerando que a seleção (roleta) tenha sorteado, nesta ordem: o 2º mais apto, o 1º mais apto, novamente o 2º mais apto e o 3º mais apto (matou o menos apto).
3. Recombine os 2 primeiros após o 3º gene (gerando 2 filhos) e os 2 últimos após o 5º gene (mais 2 filhos). Mostrar a população atual.
4. Faça a mutação (cromossomo, novo valor, gene): (1º, 1, 6º), (3º, 2, 3º), (4º, 7, 8º). Mostre a nova população.

37

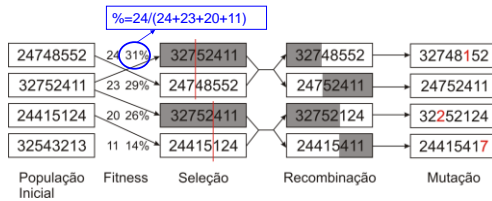
Exemplo: 8 Rainhas

Recombinação:

38

Exemplo: 8 Rainhas

- Indivíduos com maiores aptidões possuem mais chances de ser selecionados

%=24/(24+23+20+11)

População Inicial	Fitness		Seleção	Recombinação	Mutação
24748552	24	31%	32752411	32748552	32748152
32752411	23	29%	24748552	24752411	24752411
24415124	20	26%	32752411	32752124	32252124
32543213	11	14%	24415124	24415411	24415417

39

Referências

- Slides baseados em
  - Artificial Intelligence A Modern Approach (**Cap. 4.3 e 4.4**)
  - Slides de Dana Nau, do curso CMSC 421, Intro to AI (Spring 2010)